Machine Learning Engineer Nanodegree

Capstone Project

Madhumathi Prasad

May 15th, 2018

I. Definition

Project Overview

Food is an important part of everyday life.Now a days people are being conscious about the food they eat.What they eat matters in both nutrition and calorie intake.Not only people who want to improve their health are careful about the food, but many people consider to stay healthy and eating healthy food has become a part of their lifestyle.

Food recognition is a key element in the food consumption monitoring.There are few traditional techniques in recognizing the food like referring to nutrition facts or Amazon turk.Researchers have been working on food recognition using conventional approaches based on classical image features and machine learning for many years. Joutou et al. [10] created a private Japanese food dataset with 50 classes. They proposed a Multiple Kernel Learning (MKL) method using combined features including SIFT-based bag-of-features, color histogram and Gabor Texture features. An accuracy of 61.3% on their dataset was achieved. A follow-up study by Hoashi et al. [11] achieved an accuracy rate of 62.5% using the same method on an extended dataset of 85 classes. Bossard et al. [17] created an image dataset called Food-101, which contains 101 types of food images. They presented a method based on Random Forests to mine discriminative visual components and could efficiently classify with an accuracy rate of 50.8%.

In recent years, CNN is also widely used in food recognition and provides better performance than the conventional methods. Bossard et al. [17] trained a deep CNN from scratch on Food-101 dataset using the architecture of AlexNet model (proposed by Krizhevsky et al. [18]) and achieved 56.4% top-1 accuracy. They proposed a new method based on Random Forest outperforms state-of-the-art methods on food recognition. In [19], Kagaya et al. also trained CNN for food recognition and the experimental results showed that the CNN outperformed all the other baseline classical approaches by achieving an average accuracy of 73.7% for 10 classes. Myers et al. [23] presented the Im2Calories system for food recognition which extensively used CNN-based approaches. The architecture of GoogLeNet [24] was used in their work and a pre-trained model was fine-tuned on Food101. The resulting model has a top-1 accuracy of 79% on Food-101 test set.Many more research and findings are stated in the paper,Food/Non-food Image Classification and Food Categorization using Pre-Trained GoogLeNet Model Evaluation by Ashutosh Singla et al.[1].

Problem Statement

How to derive the food information (e.g., food type) from food image effectively and efficiently remains a challenging and open research problem. The goal of the project is to classify food image using CNN.Our model should be able to predict the likelihood of the food image and classify them under the provided categories.Basically it would be a multi-class classification problem in terms of machine learning.

As Deep learning techniques have been recognised as one of the best classification problems,in this project we will start with basic CNN technique by adding conv2D, max pooling and dropout layers.Would propose to try transfer learning with pre-trained weights.We have few architecture like RESNET ,VGG-16 and inceptionv3 pretrained on imagenet readily available.Would use any of them to see if it is better in classifying the images under the provided categories or classes.

Metrics

We choose test accuracy as our evaluation metric when comparing different models. We would be reporting the test accuracy numbers to two decimal places.The model should be able to predict the images under the provided classes.Here Accuracy is calculated as counting the number of predictions that is equal to the labels, then divide it by the total number of predictions.Accuracy gives an estimate of how an often model is correct on its predictions, that is, how often model correctly flags an apple pie as apple pie or bread pudding as bread pudding.

As this is a multi class classification problem and we are interested to see how well a food can be recognised based on its label,we consider test accuracy to be suitable for our project.Moreover in our dataset we have balanced number of images in each class.There is no such class which has too many or too less images. This can be seen in our exploratory visualisation section.For example *,in cancer detection example with 100 people, only 5 people has cancer. Let's say our model is very bad and predicts every case as No Cancer. In doing so, it has classified those 95 non-cancer patients correctly and 5 cancerous patients as Non-cancerous. Now even though the model is terrible at predicting cancer, The accuracy of such a bad model is also 95%.*As we are not using such data,calculating accuracy is suitable for our model,which gives us sense of how many of our images were rightly predicted.

Apart from that,the model should be able to predict few images that are not in the dataset under the provided classes.It will be interesting to see how our model recognises images other than our dataset.

II. Analysis

Data Exploration

The dataset which we will be using for this project is 11 categories extracted from Food-101 [4],which has 101 food categories.Each category has around 750 images in training dataset and 250 images in test dataset.We have further separated training dataset into 7235 images and 1014 images in training and validation datasets respectively.

The food-101 is large dataset and has 1M images,it would take high computational cost and time to evaluate the whole dataset,so we extracted just 11 categories from that dataset.The first 11 categories of food-101 are used dataset for our project.They are: ‘'baklava', 'beef_tartare', 'baby_back_ribs', 'beignets', 'bruschetta', 'bread_pudding', 'beef_carpaccio', 'beet_salad', 'apple_pie', 'breakfast_burrito' and 'bibimbap'

Exploratory Visualization

Fig.1 shows the distribution of the training dataset.The training dataset has 11 classes and each class has around 650-700 images.The figure also shows that each class has enough data to train.Fig.2 shows the distribution of the validation dataset which is nearly 15% of the training dataset and is not evenly distributed.The validation data of each class ranges from 65-120 images.Fig.3 shows test dataset which is uniformly distributed with 250 images.

Fig.1

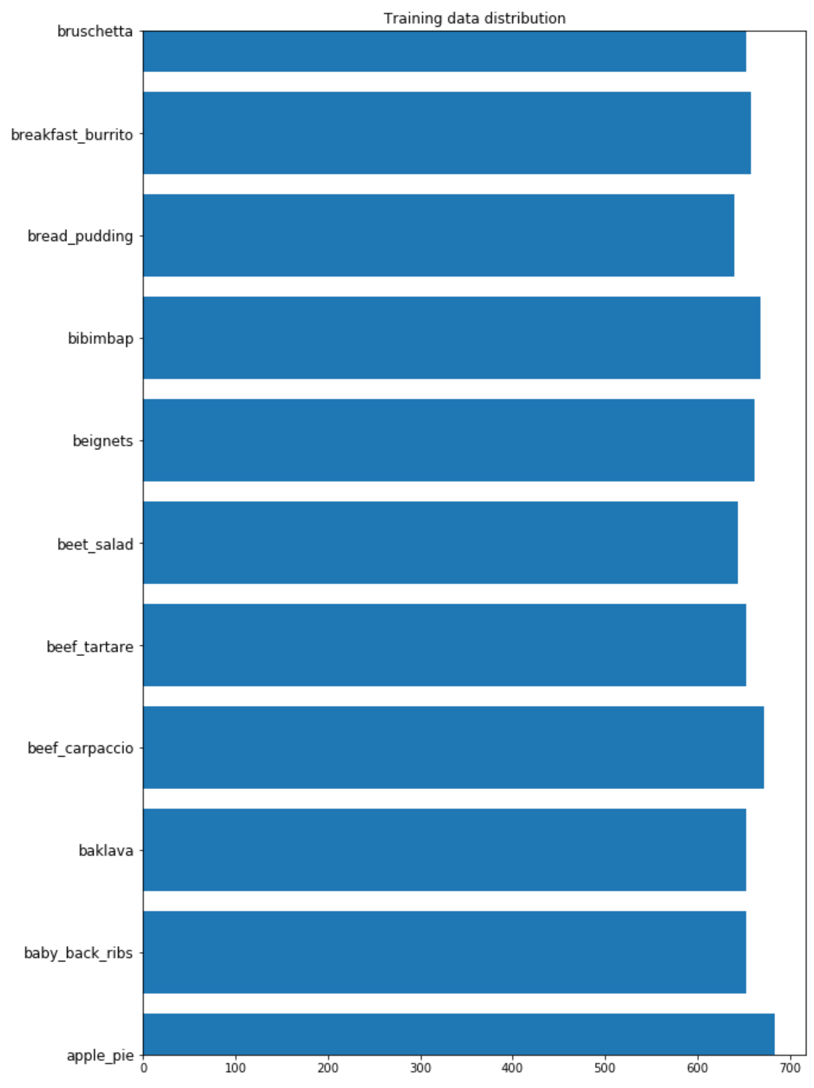


Fig.2

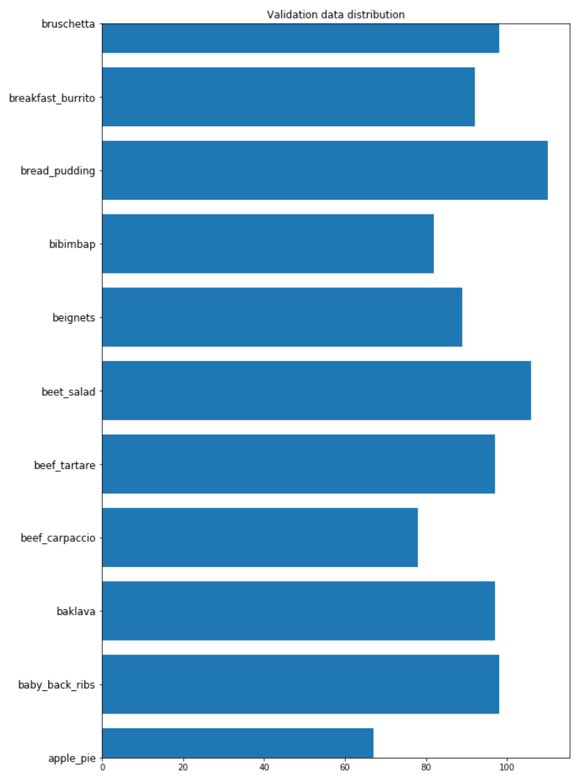
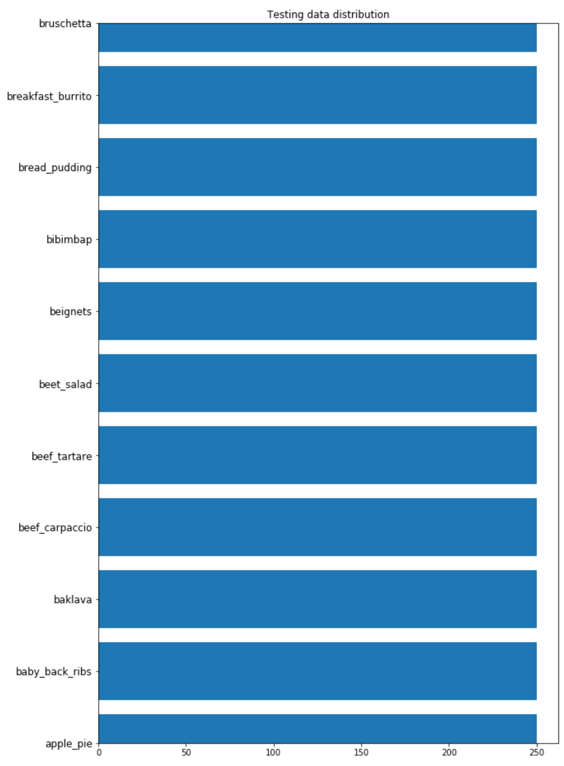


Fig.3



Algorithms and Techniques

Convolutional Neural Networks

We will be using Convolutional Neural Networks (CNNs) for this project.What is CNN? CNNs are a special kind of multi-layer neural networks. They are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they still have a loss function (e.g. SVM/Softmax) on the last (fully-connected) layer. However, ConvNet architectures make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network [8].

The Convolutional Networks are commonly made up of only three layer types: CONV, POOL (we assume Max pool unless stated otherwise) and FC (short for fully-connected). We will also explicitly write the RELU activation function as a layer, which applies elementwise non-linearity. In this section we discuss how these are commonly stacked together to form entire ConvNets.

Example Architecture: Let us say a simple ConvNet could have the architecture [INPUT - CONV - RELU - POOL - FC]. In more detail:

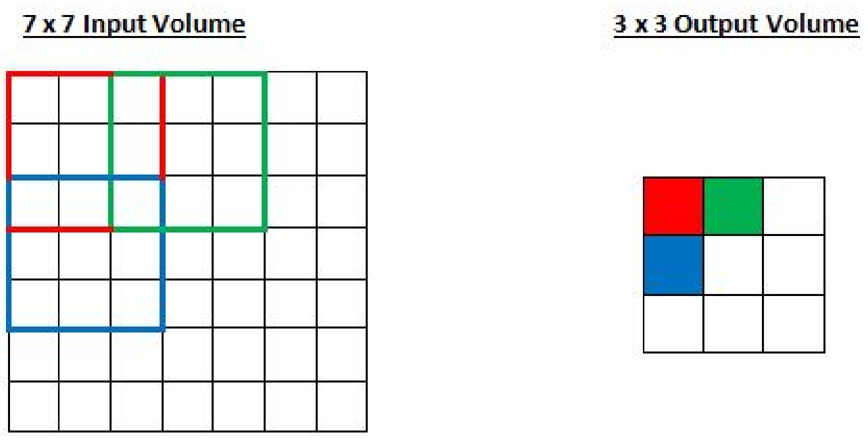
- INPUT [32x32x3] will hold the raw pixel values of the image, in this case an image of width 32, height 32, and with three color channels R,G,B.
- CONV layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume such as [32x32x12] if we decided to use 12 filters.
- RELU layer will apply an elementwise activation function, such as the max(0,x) thresholding at zero. This leaves the size of the volume unchanged ([32x32x12]).
- POOL layer will perform a downsampling operation along the spatial dimensions (width, height), resulting in volume such as [16x16x12].
- FC (i.e. fully-connected) layer will compute the class scores, resulting in volume of size [1x1x10], where each of the 10 numbers correspond to a class score, such as among the 10 categories of CIFAR-10. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume.

We now describe the hyperparameters and their connectivities to the layers.

Stride and Padding

Stride controls how the filter convolves around the input volume. The filter convolves around the input volume by shifting one/two, .. unit at a time. Stride is normally set in a way so that the output volume is an integer and not a fraction. Let’s look at an example. Let’s imagine a 7 x 7 input volume, a 3 x 3 filter (Disregard the 3rd dimension for simplicity), and a stride of 2. So, as you can see, the receptive field is shifting by 2 units now and the output volume shrinks as well. Notice that if we tried to set our stride to 3, then we’d have issues with spacing and making sure the receptive fields fit on the input volume. Normally, programmers will increase the stride if they want receptive fields to overlap less and if they want smaller spatial dimensions.

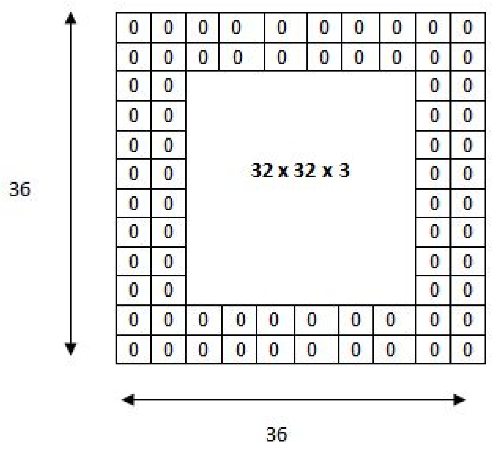
Fig.4https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/



Padding

What happens when you apply three 5 x 5 x 3 filters to a 32 x 32 x 3 input volume? The output volume would be 28 x 28 x 3. Notice that the spatial dimensions decrease. As we keep applying conv layers, the size of the volume will decrease faster than we would like. In the early layers of our network, we want to preserve as much information about the original input volume so that we can extract those low level features. Let’s say we want to apply the same conv layer but we want the output volume to remain 32 x 32 x 3. To do this, we can apply a zero padding of size 2 to that layer. Zero padding pads the input volume with zeros around the border. If we think about a zero padding of two, then this would result in a 36 x 36 x 3 input volume.

Fig. 5:



The input volume is 32 x 32 x 3. If we imagine two borders of zeros around the volume, this gives us a 36 x 36 x 3 volume. Then, when we apply our conv layer with our three 5 x 5 x 3 filters and a stride of 1, then we will also get a 32 x 32 x3 output volume.

Input and Output size

The formula for calculating the output size for any given conv layer is

input----------filter----------output

n ×× n----*----f ×× f --------[n+2p−fs+1]×[n+2p−fs+1][n+2p−fs+1]×[n+2p−fs+1]

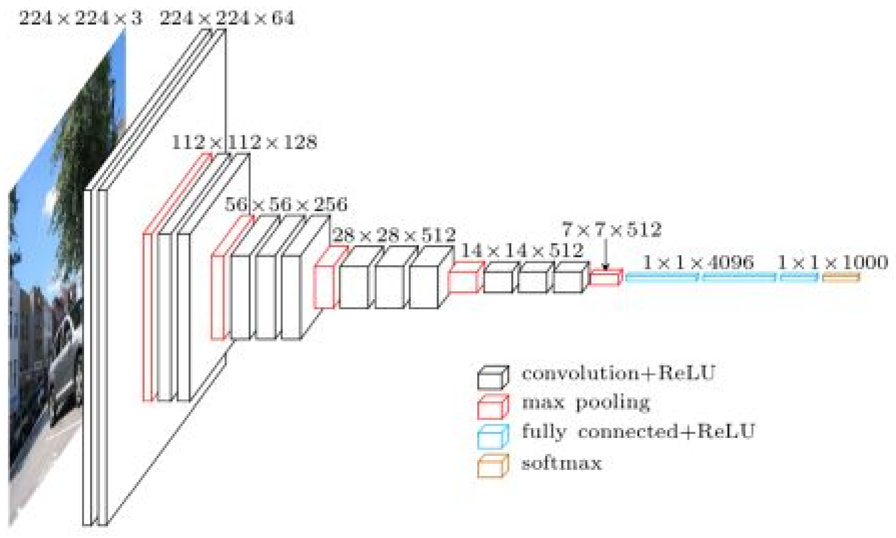
where n is input size for example 32×32×1, p is padding, f number of filters and s is stride.

There are several architectures in the field of Convolutional Networks,We will be using VGG16 for this project.

VGG16 Architecture

The runner-up in ILSVRC 2014 was the network from Karen Simonyan and Andrew Zisserman that became known as the VGGNet. Its main contribution was in showing that the depth of the network is a critical component for good performance. Their final best network contains 16 CONV/FC layers and, appealingly, features an extremely homogeneous architecture that only performs 3x3 convolutions and 2x2 pooling from the beginning to the end. A downside of the VGGNet is that it is more expensive to evaluate and uses a lot more memory and parameters (140M). Most of these parameters are in the first fully connected layer, and it was since found that these FC layers can be removed with no performance downgrade, significantly reducing the number of necessary parameters.

Fig.6 VGG16 architecture



Transfer learning

Now, a common misconception in the DL community is that without a Google-esque amount of data, you can't possibly hope to create effective deep learning models. While data is a critical part of creating the network, the idea of transfer learning has helped to lessen the data demands. **Transfer learning** is the process of taking a pre-trained model (the weights and parameters of a network that has been trained on a large dataset by somebody else) obtained bottleneck features or "fine-tuning" the model with your own dataset. The idea is that this pre-trained model will act as a feature extractor. You will remove the last layer of the network and replace it with your own classifier (depending on what your problem space is).

You then freeze the weights of all the other layers and train the network normally (Freezing the layers means not changing the weights during gradient descent/optimization).

Let’s investigate why this works. Let’s say the pre-trained model that we’re talking about was trained on ImageNet (For those that aren’t familiar, ImageNet is a dataset that contains 14 million images with over 1,000 classes). When we think about the lower layers of the network, we know that they will detect features like edges and curves. Now, unless you have a very unique problem space and dataset, network is going to need to detect curves and edges as well. Rather than training the whole network through a random initialization of weights, we can use the weights of the pre-trained model and focus on the more important layers (ones that are higher up) for training.

Benchmark

The benchmark model would be a project from the stanford university,Deep Dish : Deep Learning for Classifying Food Dishes by Abhishek Goswami Microsoft Redmond, WA and Haichen Liu Dropbox Seattle, WA .[6].The model has classified food into 20 categories,which is a different dataset than what we would be using.The model used Transfer Learning (fine tuning a VGG model) with test accuracy of 0.45.This seems to be attainable test accuracy.

III. Methodology

Data Preprocessing

Data generators are used to generate and augment images

- Generate augmented images from current images using the Keras utility ImageDataGenerator to be used to train models.
- rescale=1./255:I had to normalize the values to span from 0 to 1., which was achieved by this scaling

- width_shift_range=0.1: range in which image was randomly translated vertically
- height_shift_range=0.1: range in which image was randomly translated horizontally
- zoom_range=0.2: range in which image was zoomed at randomly
- fill_mode='nearest': this was the method with which newly introduced pixels were filled out
- Resize pictures to height and width of 224px.

Preprocessing the images

- When using TensorFlow as backend, Keras CNNs require a 4D array also referred as a 4D tensor as input, with shape (nb_samples,rows,columns,channels),
- where nb_samples corresponds to the total number of images (or samples), and rows, columns, and channels correspond to the number of rows, columns, and channels for each image, respectively.
- The path_to_tensor function takes a string-valued file path to a color image as input and returns a 4D tensor suitable for supplying to a Keras CNN. The function first loads the image and resizes it to a square image that is 224×224 pixels. Next, the image is converted to an array, which is then resized to a 4D tensor. In this case, since we are working with color images, each image has three channels. Likewise, since we are processing a single image (or sample), the returned tensor will always have shape(1,224,224,3).
- The paths_to_tensor function takes a numpy array of string-valued image paths as input and returns a 4D tensor with shape (nb_samples,224,224,3).
- Here, nb_samples is the number of samples, or number of images, in the supplied array of image paths. It is best to think of nb_samples as the number of 3D tensors (where each 3D tensor corresponds to a different image) in your dataset!

Implementation

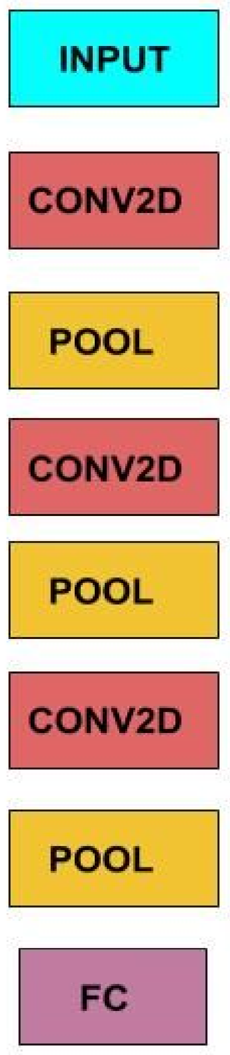
Project Requirements

- Python 3.5
- Tensorflow 1.4.0
- Keras 2.0 with tensorflow as backend.

- GPU(optional)

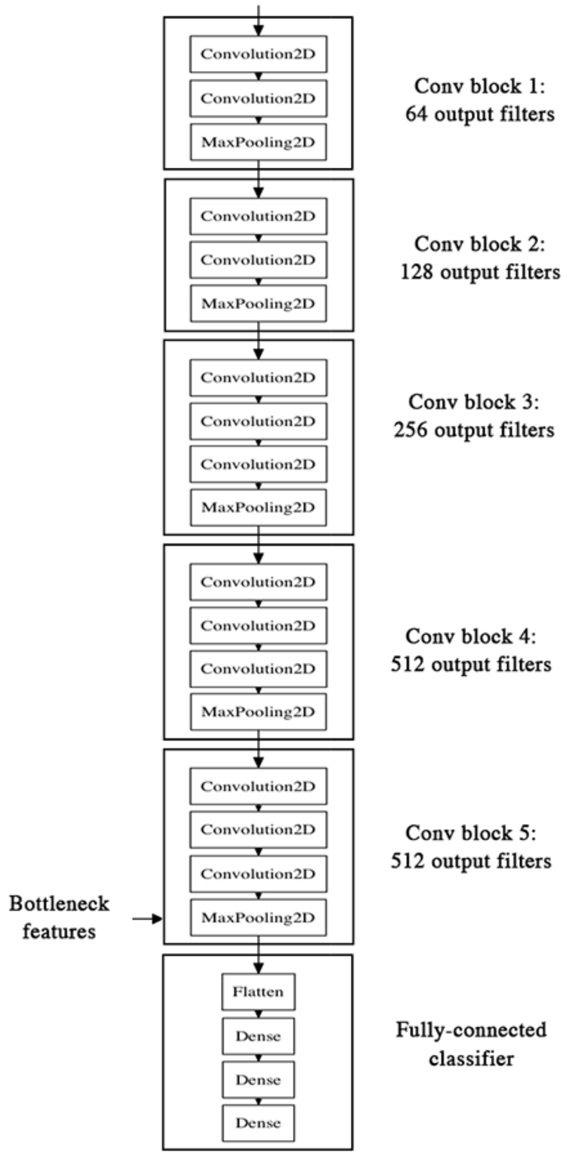
For the implementation I have chosen Keras. Keras is a neural network library for Theano and TensorFlow written in Python. For the activation functions rectified linear units are used, except for the final output neuron which was sigmoid. A CNN model with following layers was built with layers as in Fig.7.And the loss function I have used is categorical cross entropy with Rmsprop as optimiser.

Fig.7 Layers of the CNN model used in the project.



The CNN model used had accuracy of 55%.So decided to choose another architecture VGG16.But building a network from scratch takes time and also increases cost,So I did a transfer learning by obtaining bottleneck features of VGG16 pretrained on Imagenet.I used GPU to obtain the bottleneck features,Which reduced my computational time.It was taking hours to compute on my 4GB CPU.The model uses the the pre-trained VGG-16 model as a fixed feature extractor, where the last convolutional output of VGG-16 is fed as input to our model.

Fig 8. Bottleneck features and the FC layer



Our strategy will be as follow: we will only instantiate the convolutional part of the model, everything up to the fully-connected layers. We will then run this model on our training and validation data once, recording the output (the "bottleneck features" from the VGG16 model: the last activation maps before the fully-connected layers) in two numpy arrays. Then we will train a small fully-connected model on top of the stored features.

The reason why we are storing the features offline rather than adding our fully-connected model directly on top of a frozen convolutional base and running the whole thing, is computational efficiency. Running VGG16 is expensive, especially if you're working on CPU, and we want to only do it once. Note that this prevents us from using data augmentation.

First time ,I added a global average pooling layer,a dropout of 0.5 and a fully connected layer, where the latter contains 11 nodes, one node for each food category and is equipped with a softmax.I trained the model with RMSprop optimiser.I got test accuracy: 57.05%,which was slightly better than the basic CNN model.

Refinement

Next I optimised the model,I added a dense layer with relu activation,this time the test accuracy went upto 61.74%.Then I tried the same model with a different adam optimiser.A test accuracy of 63.41% was obtained.The table below sums up the architecture,optimiser and corresponding accuracy obtained.

	Architecture	optimiser	Accuracy%
1.	Basic CNN	rmsprop	55
2.	Pretrained VGG16	rmsprop	57.05
3.	Pretrained VGG16 with added Dense layer	rmsprop	61.74
4.	Pretrained VGG16 with added Dense layer	Adam	63.41

IV. Results

Model Evaluation and Validation

The final model is a pre-trained VGG16 model with a global average pooling layer,a relu activated dense layer with 256 nodes ,a dropout of 0.5 and a fully connected layer, where the latter contains one node for each food category and is equipped with a softmax.The model gets a slight increase in test accuracy by changing the optimiser to ‘Adam’.

The model achieves a accuracy of 63.41% we can use some other architecture and see if this value goes up.Another thing would be fine tuning the last fully connected layer.

The graphs below showing loss and accuracy of train and test dataset.The loss of training dataset decreases over epoch and accuracy increases over the epoch.The validation loss and accuracy constantly increases and decreases over epochs.

Fig. 9

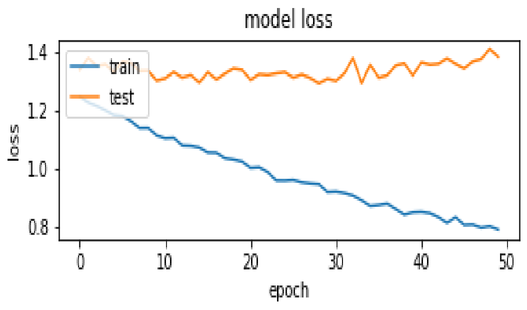


Fig. 10.

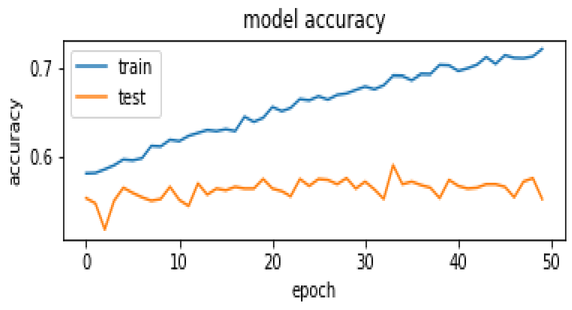
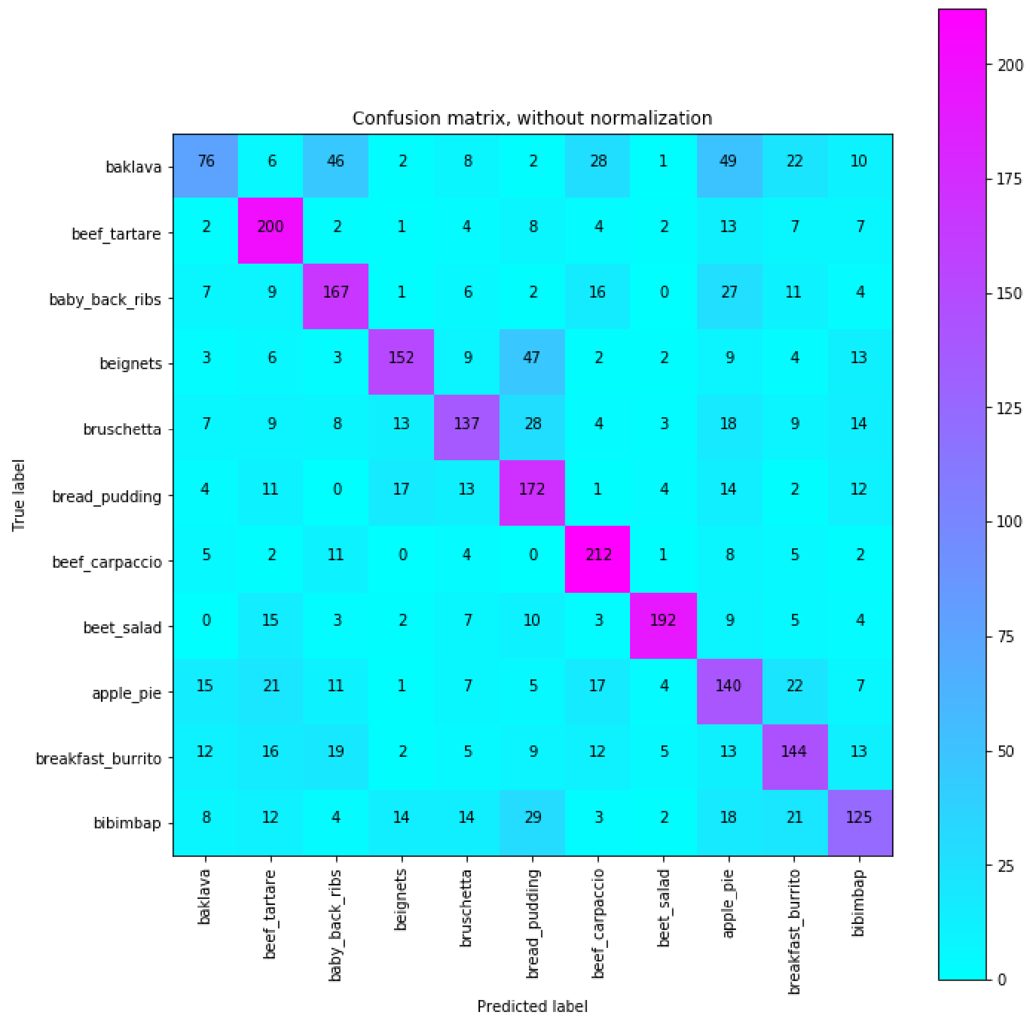


Fig. 11 Confusion Matrix.



True label \ Predicted label	baklava	beef_tartare	baby_back_ribs	beignets	bruschetta	bread_pudding	beef_carpaccio	beet_salad	apple_pie	breakfast_burrito	bibimbap
baklava	76	6	46	2	8	2	28	1	49	22	10
beef_tartare	2	200	2	1	4	8	4	2	13	7	7
baby_back_ribs	7	9	167	1	6	2	16	0	27	11	4
beignets	3	6	3	152	9	47	2	2	9	4	13
bruschetta	7	9	8	13	137	28	4	3	18	9	14
bread_pudding	4	11	0	17	13	172	1	4	14	2	12
beef_carpaccio	5	2	11	0	4	0	212	1	8	5	2
beet_salad	0	15	3	2	7	10	3	192	9	5	4
apple_pie	15	21	11	1	7	5	17	4	140	22	7
breakfast_burrito	12	16	19	2	5	9	12	5	13	144	13
bibimbap	8	12	4	14	14	29	3	2	18	21	125

I calculated the confusion matrix with non-normalised plot of test dataset,the figure above shows that beef_carpaccio and beef_tartare are the most accurately predicted classes.Baklava is misclassified with many other classes,may be because of the texture and color.

Justification

The benchmark model mentioned in the Benchmark section uses a Transfer Learning (fine tuning a VGG model) with test accuracy of 45%.The model has classified food into 20 categories,which is a different dataset than what we we used.Our test accuracy using Transfer Learning(Bottleneck Features) has achieved a test accuracy of 63.41%

V. Conclusion

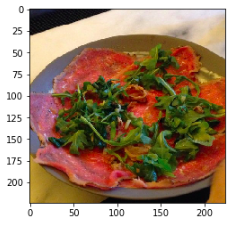
Free Form Visualization

I've predicted few correct Predict labels and few incorrect predicted labels at random.

Correct Predicted labels are

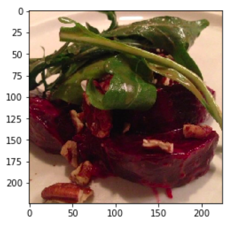
Actual : beef_carpaccio

Predicted: beef_carpaccio



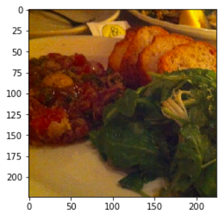
Actual : beet_salad

Predicted : beet_salad



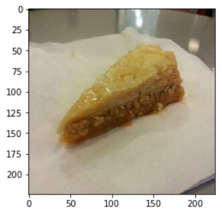
Actual : beef_tartare

Predicted : beef_tartare



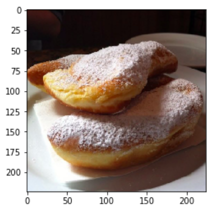
Actual : baklava

Predicted : baklava



Actual : beignets

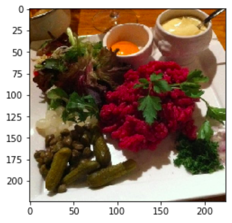
Predicted: beignets



Incorrect Predicted labels are

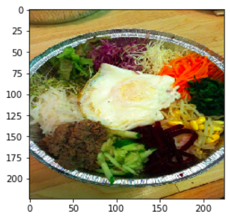
Actual : beef_tartare

Predicted: beet_salad



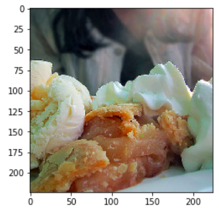
Actual : bibimbap

Predicted : beet_salad



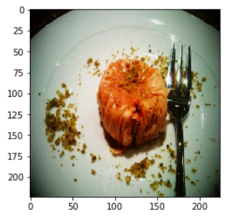
Actual : apple_pie

Predicted : baklava



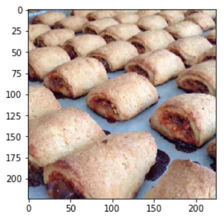
Actual : baklava

Predicted : beef_tartare



Actual : apple_pie

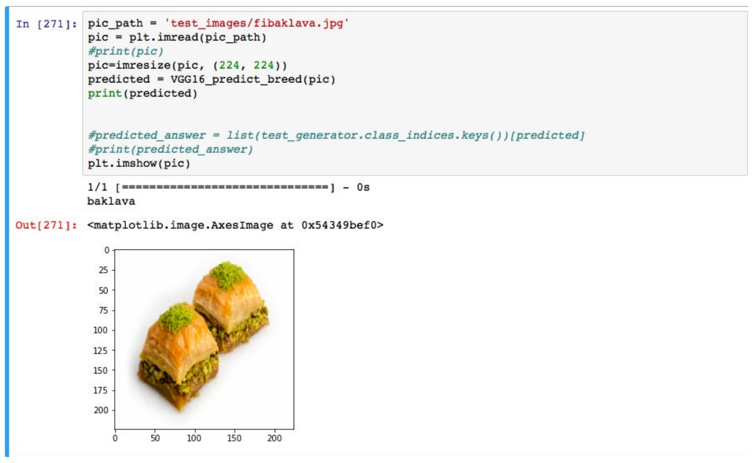
Predicted: beignets



Apple pie is incorrectly predicted as baklava and beignets.This could be maybe because the model is learning from the color.The beef_tartare is also misclassified as beet salad and the reason here could also be the red color of the beet salad.

As I mentioned in my metrics section,I downloaded few images from internet.The Fig 12 shows baklava is predicted as baklava while Fig. 13 shows bread pudding is misclassified as baklava.

Fig. 12 baklava predicted as baklava



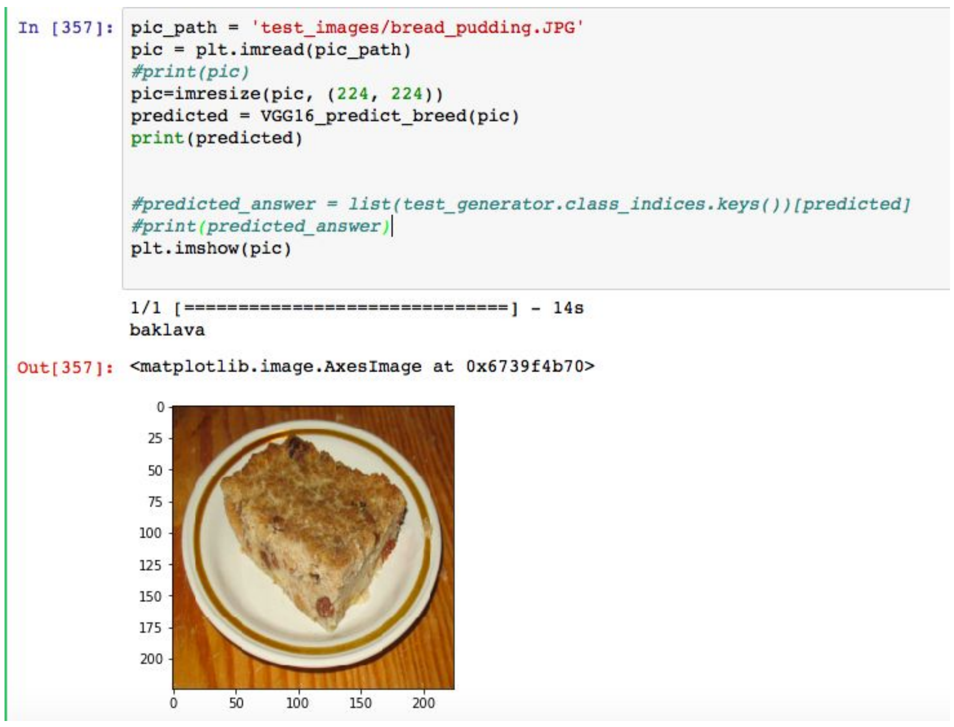
```
In [271]: pic_path = 'test_images/fibaklava.jpg'
          pic = plt.imread(pic_path)
          #print(pic)
          pic=imresize(pic, (224, 224))
          predicted = VGG16_predict_breed(pic)
          print(predicted)


          #predicted_answer = list(test_generator.class_indices.keys())[predicted]
          #print(predicted_answer)
          plt.imshow(pic)

          1/1 [==============================] - 0s
          baklava

Out[271]: <matplotlib.image.AxesImage at 0x54349bef0>
```

Fig. 13 bread pudding is misclassified as baklava.



```
In [357]: pic_path = 'test_images/bread_pudding.JPG'
          pic = plt.imread(pic_path)
          #print(pic)
          pic=imresize(pic, (224, 224))
          predicted = VGG16_predict_breed(pic)
          print(predicted)


          #predicted_answer = list(test_generator.class_indices.keys())[predicted]
          #print(predicted_answer)
          plt.imshow(pic)

          1/1 [==============================] - 14s
          baklava

Out[357]: <matplotlib.image.AxesImage at 0x6739f4b70>
```

Reflection

Few years before Image recognition used to be a difficult task, but there are efficient methods to approach such problems. Deep learning is capable of building up a hierarchy of abstractions that makes it possible to identify complex inputs (i.e. images), and in this project CNN is been used.

There were three major parts for the project. The first was data collection, the second was building the model and the third was transfer learning.The original dataset had 1M images with 101 classes but thinking it would take more time and high computational cost,the dataset is reduced to 11 classes with 10099 images.Given that collected dataset is reduced, an important part in this project is the use of the data augmentation utility from Keras to help to prevent overfitting and improve generalization.It would have been difficult without Keras to do data augmentation.It was interesting to use ImageDataGenerator for real-time data augmentation in Keras to augment images and check how the data got augmented.

After this, building the different models attempted was not particularly complex using Keras. There are a significant amount of parameters to experiment such as the type of activation functions, regularization methods, loss functions, error metrics, nodes in fully connected layers, etc., It all started from good architectures that were published by @fchollet, and build from there. It was a biggest challenge to build another CNN architecture from scratch like VGG16 or InceptionV3 to obtain high accuracy or build a sturdy model.Transfer learning helped in resolving it, by extracting the features and just adding the fully connected layer to pre trained model.Again Keras plays a major role , building powerful image classification models using very little data by @fchollet was very useful.Learning how to obtain the bottleneck features and to experience how fast it was possible to set up an architecture using little data is amazing.

Although the final method gets only an accuracy of 63% further increase in accuracy say above 95% would be desired. Other architectures like InceptionV3 and learning with more categories would be desired.The difficulty here was the time to obtain the bottleneck features,thanks to GPU ,it saved lot of time.The interesting thing was to see how fast the model fitting was done using bottleneck features even on CPU. Additional categories could be used to warranty model to generalize well enough in largely different environments.This would definitely increase the accuracy well above 95%.

In future will be experimenting and working more on deep learning would definitely try to work with more computational power confidently.Due to lack of time and computational cost trying the same model with more architectures like InceptionV3 and RESNET50 is highly challenging.Also with more food categories it would be interesting to learn how each food is being classified.

Improvement

Further improvements as already mentioned, adding additional time,high computational power and additional categories would help in generalization and increasing accuracy.A test accuracy more than 95% would be considered as an achievement.Another further area to expand the project, is building a mobile app where a smartphone camera can detect food pictures and classify the food.

References:

1. Food/Non-food Image Classification and Food Categorization using Pre-Trained GoogLeNet Model Evaluation Metrics.In:https://infoscience.epfl.ch/record/221610/files/madima2016_food_recognition.pdf

2. Food Recognition using Fusion of Classifiers based on CNNs In:https://pdfs.semanticscholar.org/9c1c/4e6ab238f34bef0348b04bfed78cda11e9ed.pdf

3. https://www.analyticsvidhya.com/blog/2017/06/transfer-learning-the-art-of-fine-tuning-a-pre-trained-model.

4. Dataset: https://www.vision.ee.ethz.ch/datasets_extra/food-101/

5. Bossard, L., Guillaumin, M., Van Gool, L.: Food-101 - Mining discriminative components with random forests.

6. http://cs231n.stanford.edu/reports/2017/pdfs/6.pdf

7. Food Image Recognition by Using Convolutional Neural Networks (CNNs).In:https://arxiv.org/pdf/1612.00983.pdf

8. http://cs231n.github.io/convolutional-networks/

9. https://www.pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/

10. Taichi Joutou and Keiji Yanai. A food image recognition system with multiple kernel learning. In 2009 16th IEEE International Conference on Image Processing (ICIP), pages 285–288, Nov 2009.

11. H. Hoashi, T. Joutou, and K. Yanai. Image recognition of 85 food categories by feature fusion. In Multimedia (ISM), 2010 IEEE International Symposium on, pages 296–301, Dec 2010.

12. https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/

13. https://medium.com/greyatom/performance-metrics-for-classification-problems-in-machine-learning-part-i-b085d432082b

14. The Udacity Dog breed classifier project.

15. https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

16. https://gist.github.com/fchollet/f35fbc80e066a49d65f1688a7e99f069

17. Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101 – mining discriminative components with random forests. In European Conference on Computer Vision, 2014.

18. Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, Advances in Neural Information Processing Systems 25, pages 1097–1105. Curran Associates, Inc., 2012.

19. Hokuto Kagaya, Kiyoharu Aizawa, and Makoto Ogawa. Food detection and recognition using convolutional neural network. In Proceedings of the 22Nd ACM International Conference on Multimedia, MM ’14, pages 1085–1088, New York, NY, USA, 2014. ACM

20. Austin Myers, Nick Johnston, Vivek Rathod, Anoop Korattikara, Alex Gorban, Nathan Silberman, Sergio Guadarrama, George Papandreou, Jonathan Huang, and Kevin Murphy. Im2calories: towards an automated mobile vision food diary. In ICCV, 2015.

21. Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In Computer Vision and Pattern Recognition (CVPR), 2015.